САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ТЕОРИИ УПРАВЛЕНИЯ

Горбатюк Анна Витальевна

Магистерская диссертация

Визуализация медицинских данных средствами виртуальной реальности

Направление 01.04.02

Прикладная математика и информатика

Магистерская программа прикладная математика и информатика в задачах медицинской диагностики

Научный руководитель,
кандидат физ.-мат. наук,
доцент
Плоских В.А.

Санкт-Петербург
2018

Содержание

Введение

Визуализация медицинских данных занимает важное место в современной медицине и биологии. Наука не стоит на месте, поэтому существует постоянный рост темпов исследований. С каждым днем наблюдается рост объема производимых трехмерных данных, которые нуждаются в методах трехмерной визуализации, обработке, реконструкции и анализе.

С помощью методов медицинской визуализации можно без оперативного вмешательства воссоздать внутренние структуры тела человека, пронаблюдать за функцией некоторых органов или тканей. Это позволяет диагностировать заболевания, принять необходимы меры при планировании лечения пациента и хирургическом вмешательстве. Область применения методов визуализации широка: исследование и разработка в области приборостроения, получение изображений, биомедицинская инженерия, медицинская физика и информатика.

Долгое время в медицине использовались двухмерные методы визуализации, которые до сих пор находят свое применение на практике. Однако после появления 3D томографии, которая предлагала низкую стоимость, высокое разрешение, меньшие дозы радиации и более качественный результат, стали все чаще использоваться методы трехмерной визуализации. Возможность преобразования нескольких сотен двухмерных изображений в интерактивную трехмерную модель стало настоящим прорывом. Возможно не только вращать, перемещать объект, но и увидеть отдельный срез, если это необходимо. Такая модель позволяет врачам более точно диагностировать заболевания и лучше проводить планирование лечения пациента.

Не так давно появилась новая технология, которая носит название - виртуальная реальность. Это созданная техническими средствами трехмерная

среда, с которой пользователь может взаимодействовать, полностью или частично погружаясь в нее. Виртуальная реальность имитирует как воздействие, так и реакции на воздействие. При помощи, например, очков виртуальной реальности, человек может увидеть искусственную сцену прямо перед собой, а также взаимодействовать с ней в реальном времени при помощи геймпада или других устройств для манипуляции.

На данный момент постепенно виртуальная реальность находит свое применение в жизни людей. Существует много игр, виртуальных гидов и тренажеров, использующих данную технологию. Но до сих пор в задачах медицинской визуализации она не использовалась. Данная работа будет посвящена визуализации медицинских данных с использованием средств виртуальной реальности. Программное обеспечение будет носить демонстрационный характер, таким образом его можно будет применять в научных исследованиях, промышленных задачах, а также в учебных целях.

Чтобы получить изображения в виртуальной реальности, необходимо будет создать программное обеспечение, которое в свою очередь будет способно читать медицинские данные не только в формате raw data, но и работать с dicom форматом, преобразовывать данные в трехмерную текстуру при помощи шейдеров, получать трехмерную модель объекта методом проекций максимальных интенсивностей, выводить на экран два изображения для левого и правого глаза отдельно, обеспечивать взаимодействие с объектом при помощи мыши и геймпада, способное при помощи шейдеров создавать текстуры, использующиеся как палитры для раскрашивания объекта, изменения яркости и контраста. Результат данного программного обеспечения – полноэкранное изображение, поделенное на две половины с изображением для каждого глаза соответственно.

Для получения изображения в виртуальной реальности можно использовать драйвер Trinus PSVR, который преобразует имеющееся изображение в то, что можно будет увидеть в очках виртуальной реальности.

Существует альтернативная возможность. Это использование дополнительно сторонней библиотеки OpenVR и Stream VR перед TrinusVR. Отличие состоит в том, что в первом случае необходимо настраивать вручную расстояние между глазами и положение камеры для каждого глаза, во втором случае с этими задачами помогает справиться OpenVR. Более подробно с особенностями реализации можно ознакомиться в тексте данной работы.

Постановка задачи

Целью данной работы является создание программного обеспечения, способного визуализировать медицинские данные средствами виртуальной реальности. Для достижения этой цели необходимо учесть ряд требований, таких как:

1) Возможность работы с медицинскими данными форматов raw data и dicom.

2) Построение объемного изображения по входным данным.

3) Построение 3D изображения для каждого глаза отдельно.

4) Возможность изменять положение модели при помощи мыши и геймпада.

5) Возможность изменять палитру, яркость, контраст.

6) Преобразование 3D изображения для виртуальной реальности.

Обзор литературы

[1] - статья, в которой можно узнать больше в общем о медицинской визуализации, о ее методах и направлениях, в которых она находит свое применение, а также историю развития. Чтобы немного глубже погрузиться именно в визуализацию трехмерных данных можно прочесть диссертацию [2], в которой автор рассказывает о методах высокопроизводительной визуализации и анализе трехмерных данных в задачах медицины и биологии.

В статьях [3] и [4] можно узнать о том, что же такое виртуальная реальность, об истории данной технологии, областях использования, свойствах, классификациях и использующемся оборудовании.

Чтобы лучше разобраться в вопросах, связанных с объемным рендерингом, можно обратиться к русскоязычным источникам [5] и [6]. В них можно прочесть об этапах объемного рендеринга, найти обзоры методов, используемых для каждого этапа визуализации объема. Методы иллюстрируются примерами из различных областей медицины. На эту тему есть еще один англоязычный источник [7], который включает в себя помимо всего вышеперечисленного алгоритмы для шейдеров, архитектуры и применения объемной визуализации.

Хорошая статья на русском [8] поможет разобраться с преобразованием координат одного пространства в координаты другого пространства, а также поможет понять, в чем разница между ортогональной и перспективной проекциями. Аналогичным образом про основные матрицы трансформации можно узнать в англоязычном источнике [9], а про трансформацию в OpenGL - из англоязычного источника [10].

[11] и [12] помогут разобраться с таким понятием, как кватернионы и тем, как используя их, можно осуществить вращение объектов.

Для работы с трехмерной графикой необходимо хорошо понимать устройство и принципы работы графического конвейера. Разобраться с этим можно, если прочесть статьи [13] и [14]. Там также объяснено, зачем нужны шейдеры, какие виды шейдеров встречаются и как это связано с графическим конвейером. Чтобы написать простые шейдеры на GLSL можно обратиться к обучающим статьям [15] и [16].

Для желающих реализовать орбитальную камеру или узнать получше как она работает доступно два источника [17] и [18]. В первом рассказывается о том, как работает орбитальная камера, использующая кватернионы. Во второй статье описывается как при помощи средств, доступных в OpenGL, имитировать простую орбитальную камеру.

Русскоязычных источников, в которых бы говорилось про метод максимальных интенсивностей очень мало. В статье [19] рассказывается в общем обо всех методах интенсивностей, в чем между ними разница и раскрываются их особенности. Алгоритм метода максимальных интенсивностей можно найти в [20]. Среди англоязычных источников есть несколько хороших статей. В [21] рассказывается о применении метода проекции интенсивности локальных максимумов в задачах связанных с визуализацией сосудов. В [22] интересным образом проводится сравнение объемного рендеринга и метода проекции максимальных интенсивностей на примере задач CT. Там можно узнать какой из методов работает быстрее, в каких случаях и почему это происходит.

Если есть заинтересованность в изучении вопроса касательно Dicom формата, то можно прочитать [23].

Статья [24] будет полезна в том случае, когда необходимо будет разобраться с особенностями OpenGL.

Все медицинские данные, которые использовались в данной работе были взяты из [25].

Глава 1. Существующие аналоги.

В данной главе будут приведены примеры нескольких существующих аналогов программного обеспечения для работы с визуализацией медицинских данных с их кратким описанием.

1. VTK: Visualisation Toolkit. Это бесплатная система с открытым кодом для трехмерной компьютерной графики, обработки изображений и визуализации. Содержит C++ библиотеки и Tcl/Tk, Java и Python. Содержит набор трехмерных виджетов взаимодействия, поддерживает параллельные вычисления и интегрируется с многими базами данных через GUI инструментарий. Кросс платформенный продукт: Linux, Windows, Mac и Unix платформы.
2. XNAT:The Extensible Neuroimaging Archive Toolkit. Программная платформа с открытым исходным кодом, которая облегчает управление и исследование нейровизуализации и связанных с ней данных. Имеет защищенную базу данных и обширный веб интерфейс для пользователей. Возможен просмотр изображений локально и через соединение по сети. Язык программирования Java. Платформы: Linux, Windows, Mac и Virtual machines)
3. MIVAP:Medical Image Processing, Analysis and Visualisation. Программное обеспечение, позволяющее производить количественный анализ и визуализацию медицинских данных, полученных из Pet, MRI, CT или микроскопа. Используется для DICOM исследований, поддерживает различные форматы на чтение и запись. Основные функции: конвертация данных и визуализация. Язык программирования: Java. Платформы: Linux, Windows, Mac.
4. 3DimViewer: 3D Dicom viewer. Программное обеспечение с открытым кодом, которое отображает мультипланарные и ортогональные изображения, используется объемная и поверхностная визуализация с сегментацией данных на основе

пороговых значений. Для визуализации объема используется GPU ускорение. Язык программирования: C++. Платформы: Linux, Windows, Mac.

5. Santesoft Dicom viewer 3D. Программное обеспечение, которое позволяет просматривать Dicom данные с использованием изоповерхностей и разными техниками проекций. Выполняет двухмерный и трехмерный рендеринг. Язык программирования C++. Поддерживается только для Windows.

Приведенные примеры составляют только часть программных продуктов, которые существуют в настоящее время. Часть из них предназначена для просмотра и анализа медицинских изображений с простыми методами визуализации. Другие же содержат в себе ряд сложных компонентов и методов для более быстрой и качественной визуализации медицинских данных. Третьи же специализируются на решении очень узкого ряда задач, как например программное обеспечение под номером 2, предназначенное для решения задач нейровизуализации, для которой стандартный набор методов будет работать либо медленно, либо некачественно.

Поскольку главная задача состоит в том, чтобы использовать технологию виртуальной реальности для медицинских изображений, это значит, что требуется простота решения и для реализации можно выбрать один из простых стандартных методов визуализации. Тем не менее, стоит отметить, что аналогов программного обеспечения, которые бы использовали виртуальную реальность для решения задач медицинской визуализации найдено не было.

Глава 2. Основные понятия

2.1 Графический конвейер

Во всех современных видеокартах применяется конвейерная модель обработки данных. Это сделано для увеличения быстродействия. Графический конвейер - это совокупность аппаратных и программных средств обработки трехмерных сцен, конечным итогом работы которого является кадр, размещенный на экране монитора. Именно поэтому важно понимать, как он работает. На рисунке 2.1.1 изображена простая схема графического конвейера.

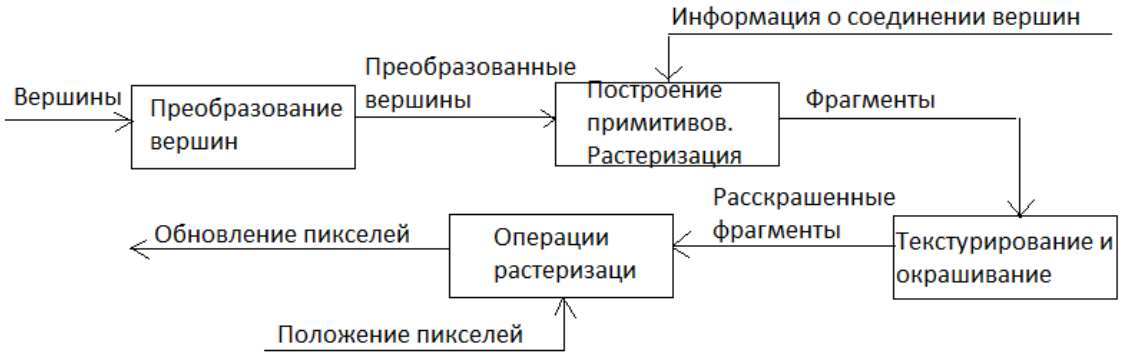

Рис. 2.1.1 Упрощенная схема графического конвейера с его состояниями и данными, которые по нему проходят.

Все точки, которые имеются в сцене проходят этапы графического конвейера. Каждая вершина имеет определенный набор атрибутов, таких как позиция, цвет, текстурные координаты, вектор нормали и т.д. На этапе преобразования вершины на вход поступают ее атрибуты, происходит трансформация позиции вершины, генерируются и преобразовываются текстурные координаты и т.д.

На этапе построения примитивов и растеризации на вход идет информация о трансформации вершины и о соединении вершин. Из этих данных осуществляется сборка геометрических примитивов. Так же происходит отсечение примитивов и могут отбрасываться грани, которые находятся сзади, определяется это по порядку следования вершин.

Растеризация – это процесс определения пикселей, покрывающих геометрический примитив.

На этапе текстурирования и окрашивания происходит интерполяция, а также последовательность математических преобразований и операций текстурирования, что определяет конечный цвет каждого фрагмента.

На последнем этапе проводится ряд по-фрагментных тестов. Например, тест отсечения, тест прозрачности, тест глубины и т.д. Таким образом определяется конечный вид, цвет и глубина фрагмента перед обновлением экранного буфера. После тестов следует смешивание – операция, которая комбинирует финальный цвет фрагмента с текущим цветом пиксела и записывает в экранный буфер.

На двух из перечисленных этапах используются вершинный (на этапе преобразования вершин) и фрагментный шейдеры (на этапе текстурирования и окрашивания). Шейдер – это компьютерная программа, предназначенная для исполнения процессорами видеокарты. (см. п. 4.3). Шейдеры составляются на одном из специализированных языков (в рамках данной задачи GLSL) и подключаются в основной код программы. Используются шейдеры в трехмерной графике для определения окончательных параметров объекта или изображения, могут включать в себя описания наложения текстур, отражения преломления, затенения и много других параметров.

2.2 Объемный рендеринг

Объемный рендеринг – это техника, используемая для получения изображения трехмерного дискретного набора данных. Под входным набором данных подразумевается множество плоских изображений - слоев, полученных, например, при компьютерной томографии или магнитно-резонансной томографии (может быть так же PET, SPECT и т.д.). Из полученной серии изображений получают объемное изображение.

Объемное изображение - это трехмерный массив размерности WxHxD, где W – ширина, H – длина, D – толщина. У каждого такого изображения необходимо знать размер воксела, ориентацию в пространстве и позицию.

Воксел – это элемент объемного изображения, содержащий значение элемента растра в трехмерном пространстве. Размер воксела указывается в метаинформации Dicom данных (см. п. 4.5). От размера воксела зависит мировая матрица (см. п. 2.3), которая отвечает за ориентацию в пространстве и масштабирование. Ориентация зависит от 3 векторов, а масштабирование - от размера воксела. Из объемного изображения получается трехмерная текстура. Она в свою очередь накладывается на параллелепипед, размеры которого можно вычислить по формулам 2.2.1, 2.2.2 и 2.2.3 соответственно.

$$w = image_{column} * voxelsize_x \tag{2.2.1}$$

где w – ширина параллелепипеда, $image_{column}$ - количество столбцов снимка, $voxelsize_x$ - размер воксела по x.

$$h = image_{row} * voxelsize_y \tag{2.2.2}$$

где h - высота параллелепипеда, $image_{row}$ - количество строк снимка, $voxelsize_y$ – размер воксела по y.

$$d = image_{count} * image_{thikness} \tag{2.2.3}$$

где d - длина параллелепипеда, $image_{count}$ – количество снимков, $image_{thikness}$ - толщина снимка.

Матрица *O*, зависящая от ориентации, выглядит следующим образом (2.2.4):

$$O = \begin{pmatrix} x1 & x2 & x3 & 0 \\ y1 & y2 & y3 & 0 \\ z1 & z2 & z3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2.4}$$

где (x1 x2 x3), (y1 y2 y3) и (z1 z2 z3) – векторы ориентации.

Матрица масштабирования выглядит так (2.2.5):

$$M = \begin{pmatrix} w/2 & 0 & 0 & 0 \\ 0 & h/2 & 0 & 0 \\ 0 & 0 & d/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.2.5)$$

где w, h и d – ширина, высота и длина параллелепипеда, вычисляющиеся по формулам 2.2.1 – 2.2.3.

Чтобы в конечном счете получить мировую матрицу(матрицу модели) , нужно использовать формулу 2.2.6, где O – матрицы ориентации и M – матрица масштабирования:

$$M_{model} = O \cdot M \quad (2.2.6)$$

Необходимо также знать позицию каждого объемного изображения, если их несколько. Так как в текущей задаче рассматривается только одно изображение, значение позиции не важно.

Получив объемное изображение, можно использовать один из методов рендеринга. Рендеринг или отрисовка – термин в компьютерной графике, обозначающий процесс получения изображения по модели с помощью компьютерной программы. Под моделью понимается описание объектов, т.е. геометрические данные, положение точки наблюдателя, информацию об освещении, степени наличия какого-то вещества и т.д.

Существует большое количество алгоритмов визуализации. Поэтому их делят на 4 группы: методы на основе растеризации, методы на основе трассировки лучей, методы на основе рейкастинга и методы на основе трассировки пути.

В данной работе используется один из методов на основе трассировки лучей, который имеет название – метод проекций максимальных интенсивностей. Он часто используют, чтобы отобразить какие-то участки с яркой интенсивностью, например, опухоли. Так же его применяют, когда

необходимо отобразить сосуды, предварительно перед этим снимки проходят предобработку. Часто используют модернизацию, когда уменьшается объем реконструкции, чтобы улучшить качество изображения. С протяженными объектами, практически постоянной интенсивности, например, ребрами, алгоритм справляется хуже.

Алгоритм довольно простой. Все пространство делится на вокселы и для каждого из них задана интенсивность. Из точки обзора проходят прямые через каждый пиксел, цвет пиксела соответствует максимальной интенсивности той части, которая пересекается с прямой. На рисунке 2.2.1 можно наблюдать прохождение луча сквозь объект.

Методы на основе растеризации не имеют эффекта перспективы. Методы на основе трассировки пути являются самыми ресурсоемкими. В методах на основе рейкастинга луч прекращает свое действие, когда достигает любого объекта сцены. В данной работе используется метод на основе трассировки лучей, в нем луч не прекращает свое действие, когда достигает объекта сцены.

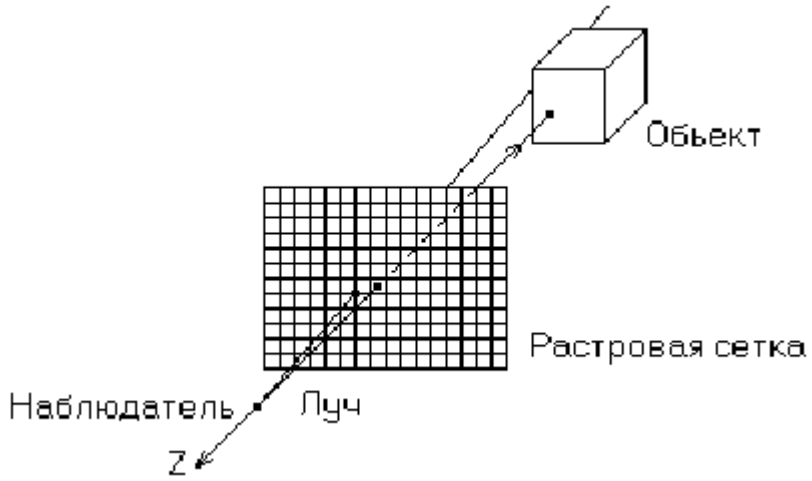

Рис.2.2.1 Прохождение луча через растровую сетку и объект.

2.3 Преобразование координат

Для того, чтобы иметь возможность отобразить пространственные объекты на экране компьютера, необходимо знать координаты объектов. В

общем случае мы имеем мировую систему координат и систему координат окна. Мировая система координат описывает истинное положение объектов в пространстве. Оконные координаты служат для вывода изображения объектов в заданной проекции. На практике, для преобразования координат одного пространства в координаты другого пространства используются несколько матриц: матрица Модели, Вида и Проекции. На рисунке 2.3.1 можно увидеть общую схему преобразования координат.

Координаты вершин объекта начинаются в локальном пространстве, как локальные координаты – это координаты объекта, измеряемые относительно точки отсчета, которая находится там, где начинается сам объект. При помощи матрицы модели они преобразуются в координаты мирового пространства – координаты объекта, которые измеряются относительно глобальной точки отсчета, единой для всех других объектов.

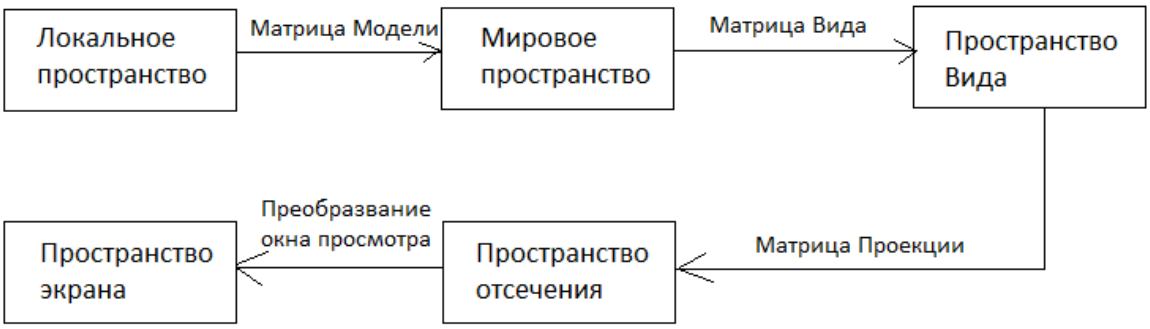

Рис. 2.3.1 Таблица преобразования координат одного пространства в другое

Матрица Модели (мировая матрица) перемещает, масштабирует и/или вращает объект для его расположения в мировом пространстве. Она отвечает за ориентацию и позицию объектов. В п. 2.1.1 рассказывалось, что матрица Модели в рамках данной задачи получалась при перемножении матрицы ориентации и матрицы масштабирования. Позиция не учитывалась, поскольку сцена состояла из одного объекта.

Следующий шаг – это преобразовать координаты мирового пространства в координаты пространства вида при помощи матрицы Вида.

Таким образом создается ощущение, будто на каждую вершину смотрят из камеры. Матрица Вида содержит в себе совокупность сдвигов, вращений и перемещений.

В рамках данной задачи, матрица Вида получалась как произведение матриц перемещения P и вращения V (2.3.1):

$$M_{view} = P \cdot V \tag{2.3.1}$$

Матрица перемещения находилась следующим образом (2.3.2):

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \mathrm{D} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.2}$$

где D – расстояние до камеры. Матрица вращения в случае управления мышью и использования правого стика геймпада вычислялось как вращение вокруг оси (x,y,z) на угол θ (2.3.3).

$$V = \begin{pmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta)\mathrm{xy} - (\sin\theta)\mathrm{z} & (1-cos\theta)\mathrm{xz} + (\sin\theta)\mathrm{y} & 0 \\ (1-cos\theta)\mathrm{yx} + (\sin\theta)\mathrm{z} & cos\theta + (1-\cos\theta)y^2 & (1-\cos\theta)\mathrm{yz} - (\sin\theta)\mathrm{x} & 0 \\ (1-cos\theta)\mathrm{zx} - (\sin\theta)\mathrm{y} & (1-cos\theta)\mathrm{zy} + (\sin\theta)\mathrm{x} & cos\theta + (1-\cos\theta)z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.3}$$

В случае нажатия на клавиши LT и RT на геймпаде, осуществляется поворот, вокруг оси z, поэтому вращение вычисляется как (2.3.4):

$$\mathrm{V}=\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ sin\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.4}$$

Матрица вращения накапливает в себе все совершенные вращения, поэтому она будет вычисляться как текущее вращение умножить справа на матрицу вращения.

Предположим, мышь из положения (x1, y1) переместилась в (x2,y2). Тогда необходимо вычислить длину вектора (2.3.5). Нормализовать

координаты (2.3.6), то есть поделить на длину, z =0. Вычислить угол по формуле 2.3.7 и подставить в 2.3.3, таким образом получится 2.3.8.

$$length = \sqrt{(x2 - x1)^2 + (y2 - y1)^2} \tag{2.3.5}$$

$$x = \frac{x2-x1}{length}, y = \frac{y2-y1}{length}, z = 0 \tag{2.3.6}$$

$$\theta = 0.1 * length \tag{2.3.7}$$

$$V = \begin{pmatrix} \cos\theta + (1-\cos\theta)x^2 & (1-\cos\theta)xy & (\sin\theta)y & 0 \\ (1-cos\theta)yx & cos\theta + (1-\cos\theta)y^2 & -(\sin\theta)x & 0 \\ -(\sin\theta)y & (\sin\theta)x & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.3.8}$$

При использовании правого стика геймпада происходят аналогичные преобразования. При нажатии LT и RT осуществляется вращение вокруг оси z. Координаты z, получаемые с геймпада, используются для вычисления угла поворота (2.3.9) и это значение подставляется в формулу 2.3.4

$$\alpha = 0.1 * z \tag{2.3.9}$$

На следующем шаге из координат пространства вида нужно получить координаты отсечения при помощи матрицы проекции. Это помогает определить видимость вершин на экране. Матрица проекции трансформирует координаты диапазона по каждой оси в нормализованные координаты устройства. Бывает двух видов: ортографическая и перспективная. В данной работе использовалась перспективная проекция для получения более реалистичной картинки благодаря эффекту перспективы. Матрицы перспективной проекции выглядит следующим образом (2.3.10):

$$M_{projection} = \begin{pmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{2.3.10}$$

где

$$f = cotangent(\frac{fovy}{2}) \quad (2.3.11)$$

$$C = -\frac{far+near}{far-near} \quad (2.3.12)$$

$$D = -\frac{2 \cdot far \cdot near}{far-near} \quad (2.3.13)$$

fovy – угол обзора в градусах, aspect – соотношение ширины к высоте, far – расстояние до дальней отсекающей плоскости от точки обзора, near – расстояние до ближайшей отсекающей плоскости от точки обзора.

После этого происходит трансформация области просмотра, которая преобразовывает координаты отсечения в область экранных координат. В конечном счете можно сказать, что координаты вершин преобразуются в координаты пространства отсечения следующим образом (формула 2.3.14):

$$V_{clip} = M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local} \quad (2.3.14)$$

где V_{local} – вектор координат вершины в локальном пространстве, M_{model}-матрица модели (мировая матрица), M_{view} – матрица вида, $M_{projection}$ – матрица проекции, V_{clip} -вектор координат вершины в пространстве отсечения.

Управление объектом осуществляется при помощи орбитальной камеры. Сначала происходит вращение объекта, а потом перемещение. Камера управляется при помощи мыши или геймпада.

Глава 3. Логика взаимодействия с VR

В данной работе будут описаны 2 возможности взаимодействия с виртуальной реальностью. На рисунке 3.1 можно увидеть первую схему взаимодействия с VR.

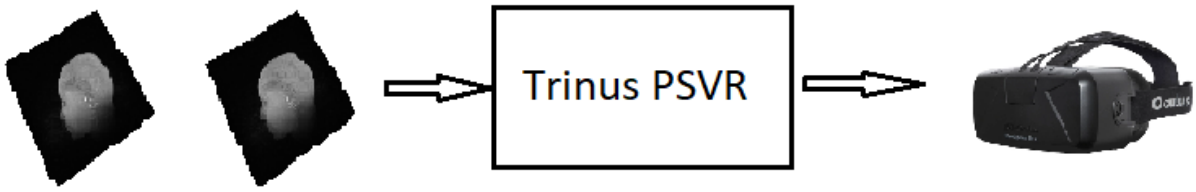

Рис.3.1 Первая схема взаимодействия с VR

Изначально создается программное обеспечение, которое обрабатывает медицинские данные и на выход выдает стереоизображение во весь экран. При помощи драйвера Trinus PSVR это изображение преобразовывается в то, что можно наблюдать в очках виртуальной реальности, т.е происходит коррекция искажений под линзы конкретной VR гарнитуры.

На рисунке 3.2 можно увидеть еще один вариант взаимодействия с VR.

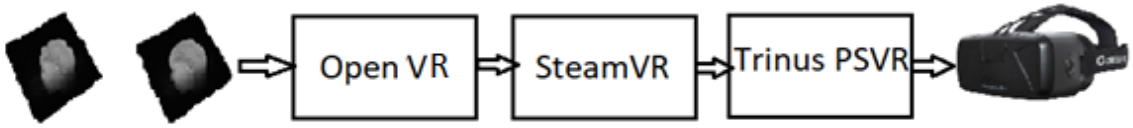

Рис.3.2 Вторая схема взаимодействия с VR

Во втором случае, аналогичным образом, необходимо программное обеспечение, которое на выходе бы выдавало полноэкранное стереоизображение. Однако теперь еще дополнительно используется сторонняя библиотека OpenVR – это программное обеспечение, разработанное для поддержки SteamVR и очков виртуальной реальности. Стоит учесть одну важную вещь - расстояние между глазами, которое составляет порядка 6-7 сантиметров, поэтому у каждого глаза своя точка обзора. Контролировать

положение камеры для каждого глаза можно при помощи матрицы Eye. Если вернуться к преобразованию координат о котором говорилось в параграфе 2.3, то с учетом матрицы Eye это будет выглядеть следующим образом (формула 3.1):

$$V_{clip} = M_{projection} \cdot M_{eye} \cdot M_{view} \cdot M_{model} \cdot V_{local} \quad (3.1)$$

где V_{local} – вектор координат вершины в локальном пространстве, M_{model}-матрица модели (мировая матрица), M_{view} – матрица вида, M_{eye} – матрица положения камеры (для каждого глаза своя матрица),$M_{projection}$ – матрица проекции, V_{clip} -вектор координат вершины в пространстве отсечения.

В случае первой схемы настраивать матрицы Eye необходимо вручную. Во втором же случае, используя возможности библиотеки OpenVR, матрицы можно получить автоматически. SteamVR используется для запуска сцены с использование драйвера Trinus PSVR, который, как и в случае с первой схемой, преобразовывает изображение, поступающее к нему из программного обеспечения в то, что можно наблюдать в очках виртуальной реальности.

При решении поставленной задачи использовалась первая схема взаимодействия с VR и значения матриц Eye устанавливались методом подбора.

Глава 4. Особенности реализации ПО

Как было сказано в главе 3 для того, чтобы получить визуализацию медицинских данных в виртуальной реальности, необходимо написать программное обеспечение, которое должно уметь работать с медицинскими данными, получать объемную визуализацию, строить стереоизображение, осуществлять управление камерой, изменением палитр, яркости и контрастности. Подробнее о том какие инструменты использовались, а также, как были реализованы те или иные моменты, будет описано ниже в данной главе.

4.1 Инструменты

При создании программного обеспечения использовались следующие инструменты:

Язык программирования – C++, GLSL.

Кроссплатформерный фреймворк – Qt.

Основные библиотеки – OpenGL, QtGamepad.

Драйвер для VR – TrinusPSVR.

4.2 Объемная визуализация при помощи шейдеров

Реализация метода максимальных проекций проходила при помощи шейдеров. О том, какие шейдеры бывают и какую функцию в программном обеспечении они играют описано в параграфе 2.1 посвященном графическому конвейеру. Шейдеры – это программы, которые выполняются видеокартой и которые встраиваются в основной код программы.

Вершинный шейдер – это программа, которая производит математические операции с вершинами, т.е изменяет параметры вершин, такие как характеристики цвета, текстурные координаты и так далее. В данном

случае определяется позиция, текстурные координаты. Ниже можно увидеть код вершинного шейдера:

```
uniform mat4 matrix;
uniform mat4 Tmatrix;
uniform mat4 Ematrix;
uniform mat4 Wmatrix;
attribute vec4 vertexAttr;
varying vec3 TextureCoord;
varying vec3 position;
void main()
{
   gl_Position = vec4(vertexAttr.xyz, 1.0)*Wmatrix*Tmatrix*Ematrix*matrix;
   TextureCoord = vec3((vertexAttr.xyz + 1.0)/2.0);
   position = vertexAttr.xyz;
}
```

Где matrix – матрица проекции, Tmatrix – матрица Вида, Ematrix – матрица положения камеры, Wmatrix – мировая матрица, vertexAttr – вектор вершин, TextureCoord – переменная, которая содержит текстурные координаты и передается в фрагментный шейдер, position – переменная, передающая в фрагментный шейдер текущую позицию.

Фрагментный шейдер – это программа, которая обрабатывает каждую видимую часть конечного изображения. В этом шейдере идет работа с освещением, тенями, отображением, текстурами и любыми другими эффектами. В результате работы получается цвет пиксела в формате RGBA (красный, зеленый, синий и альфа-канал). Ниже приведен код фрагментного шейдера, в котором используется метод максимальных интенсивностей для задания цвета пиксела:

```
uniform sampler3D textureAttr;
uniform sampler2D Stexture;
uniform float white;
uniform float black;
```

```
varying vec3 TextureCoord;
uniform vec4 eye;
varying vec3 position;

struct box
{
    vec3 min;//min corner cord
    vec3 max;
};

vec2 intersectBox(vec3 origin, vec3 dir, const box b)
{
    vec3 tMin = (b.min - origin)/dir;
    vec3 tMax = (b.max - origin)/dir;
    vec3 t1 = min(tMin,tMax);
    vec3 t2 = max(tMin,tMax);
    float tNear = max(max(t1.x,t1.y),t1.z);
    float tFar = min(min(t2.x,t2.y),t2.z);
    return vec2(tNear,tFar);
}

float trace(vec3 origin, vec3 dir, float tNear, float tFar)
{
   vec3 Mnear = origin + dir*tNear;
   vec3 Mfar = origin + dir*tFar;
   vec3 texMnear = vec3((Mnear.xyz + 1.0)/2.0);
   vec3 texMfar = vec3((Mfar.xyz + 1.0)/2.0);
   float res = 0.0;
   float step = 1.0/512.0;
   vec3 pos = texMnear;
   vec3 ray = normalize(texMfar - texMnear);
   for(int i =0; i<512; i++)
   {
      float val = texture3D(textureAttr,pos).r;
      pos = pos + step * ray;
```

```
      res = max(res,val);
   }
   return res;
}

void main()
{
   vec3 dir = normalize(position – eye.rgb);
   box square;
   square.min = vec3(-1,-1,-1);
   square.max = vec3(1,1,1);
   vec2 nf = intersectBox(eye.rgb,dir,square);
   float val = trace(eye.rgb,dir,nf.x,nf.y);
   val = (val – black)/(white – black);
   gl_FragColor  = texture2D(Stexture,vec2(val,1.0));
}
```

Где TestureAttr – трехмерная текстура, полученная из объемных данных, Stexture – двухмерная текстура, которая является палитрой, white – количество белого цвета, используется для определения контрастности и яркости, black – количество черного цвета, также применяется для контрастности и яркости, TextureCoord - переменная, которая содержит текстурные координаты, eye – вектор, который содержит координаты камеры, position – переменная, которая содержит текущую позицию.

Сначала происходит отрисовка кубика (-1,1). Из объемного изображения получаем трехмерную текстуру. Текстурные координаты вычисляются следующим образом (формула 4.2.1):

$$x' = \frac{x+1}{2}, y' = \frac{y+1}{2}, z' = \frac{z+1}{2} \quad (4.2.1)$$

где (x, y, z) - координаты вершины.

Для каждого фрагмента находится позиция камеры в локальной системе координат (eye). Далее высчитывается луч, исходящий из точки обзора. Необходимо определить ближайшее пересечение с кубом. Для этого можно использовать intersectBox. Она определяет пересечение луча с двумя

плоскостями для каждой из трех осей индивидуально. Определяет значения tNear и tFar – ближайшая точка пересечения и дальняя точка пересечения, при этом tNear должно быть меньше tFar. Вычитая из tFar значение tNear получаем луч. Находим значение максимальной интенсивности с помощью луча и по этому значению определяется цвет пиксела. Таким образом обрабатывается каждый фрагмент.

На рисунке 4.2.1 можно увидеть стереоизображение, которое получилось в результате работы данного алгоритма:

Рис. 4.2.1 Пример работы алгоритма объемной визуализации.

4.3 Палитры, яркость и насыщенность.

Полезными функциями являются изменение палитры, яркости и контрастности. Изображения могут быть различного пространственного разрешения и размера. На каждый пиксел изображения может приходиться от 1 до 24 бит. У черно-белых изображений однобайтовый пиксел, поэтому на него приходится 256 значений. При изображении в цвете используют трехбайтовые пиксели, которые содержат 16,7 мл цветов. Как правило в компьютерной томографии используют 2-байтные пикселы, количество цветов у которых 65536. Каждое изображение в зависимости от его размеров и того, с помощью чего было изображение получено (CT, PET и т.д.), имеет свою область видимости. Именно для этого и необходима возможность изменять количество белого и черного цвета. Выбирая различные палитры, можно более четко на экране увидеть медицинское изображение.

В программе на данный момент доступно 4 различные палитры: градация серого, градация красного, теплые тона и маска для PET. Переключаться между палитрами можно при помощи клавиш back и start на геймпаде. На рисунке 4.3.1 можно увидеть действие всех палитр:

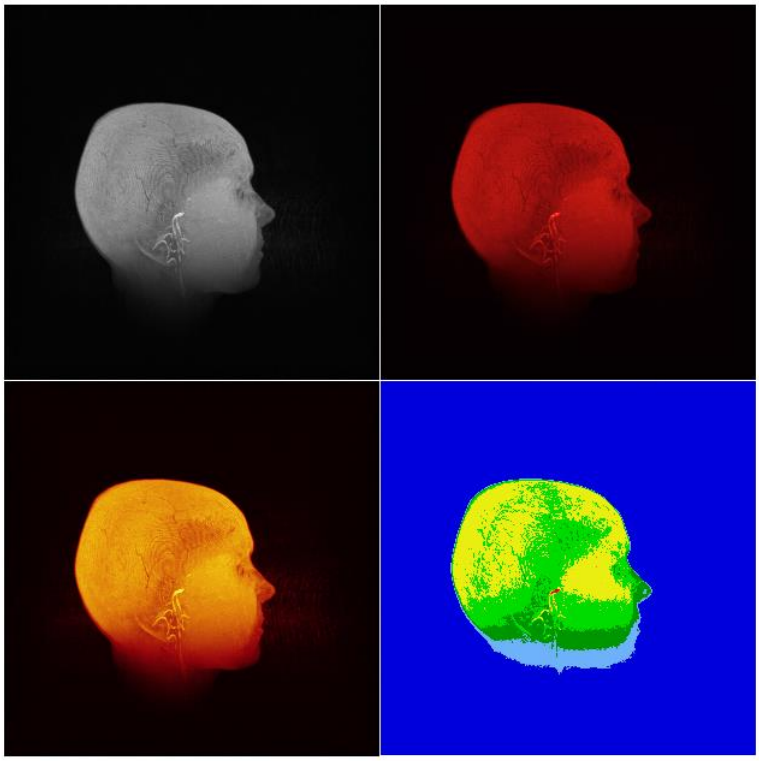

Рис. 4.3.1 Палитры «градация серого», «градация красного», «теплые тона» и «маска для PET» расположены в порядке слева на право сверху вниз.

Возможность изменять яркость и контрастность также доступна при управлении с геймпада при помощи стрелок вверх/вниз и влево/вправо соответственно. Изменение происходит при помощи регулировании черного и белого цвета изображения во фрагментном шейдере (код шейдеров приведен в параграфе 4.2) по следующей формуле (4.3.1):

$$val = \frac{val - black}{white - black} \quad (4.3.1)$$

где val – значение, полученное при трассировке лучом, black и white – значения черного и белого цвета соответственно (в промежутке от 0 до 1). В случае изменения яркости, black и white увеличиваются или уменьшаются равномерно. В случае изменения контрастности – в противофазе.

На рисунке 4.3.2 можно увидеть, как выглядит изображение до, а на рисунке 4.3.3 - как изображение изменилось после настроек яркости и контрастности:

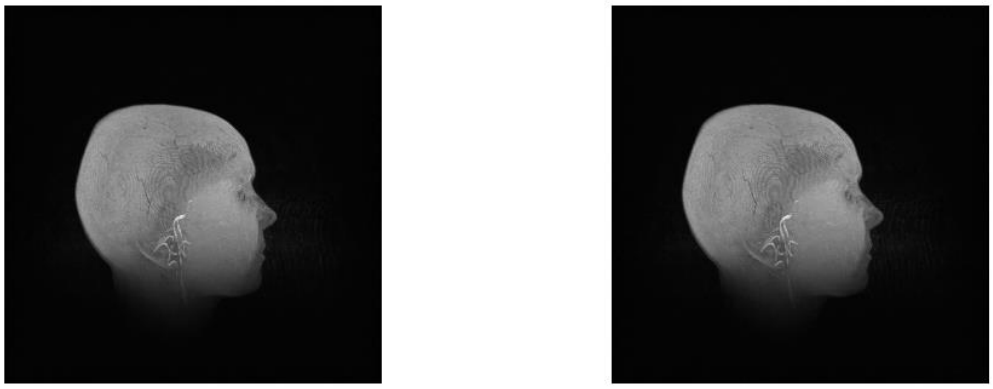

Рис.4.3.2 Стандартные настройки изображения.

Рис.4.3.3 Измененные яркость и контраст.

4.4 Чтение медицинских данных

Одно из главных требований программного обеспечения – чтение медицинских данных. Возможность читать не только данные в формате raw data, но и реализация взаимодействия с dicom данными.

Dicom - медицинский стандарт хранения, передачи и визуализации цифровых медицинских изображений и документов обследованных пациентов. Dicom файл – это объектный файл с теговой организацией для представления кадра изображения и сопровождающей или управляющей информации. Для того, чтобы отобразить такие данные, необходимо сначала

при помощи MRIConvert преобразовать данные в один из удобных форматов, который будет содержать файл с метаинформацией и raw файл с изображениями. Это может быть NIFTI, Analyze, Meta Image. Для удобства, создается собственный документ, в который помещаются лишь необходимые параметры для объемной визуализации, такие как количество строк и столбцов снимка, количество снимков, размеры воксела, толщина снимков, 3 вектора ориентации и т.д. Прочитав данные такого типа можно создать трехмерную текстуру.

На рис. 4.4.1 показан результат объемной визуализации мозга человека, полученной из PET снимков. По PET снимкам было также восстановлено туловище человека (рис. 4.4.2). На рис. 4.4.3 и 4.4.4 можно увидеть трехмерные изображения туловища человека, полученные из двух разных наборов CT снимков.

Рис 4.4.1 Изображение мозга (PET снимки).

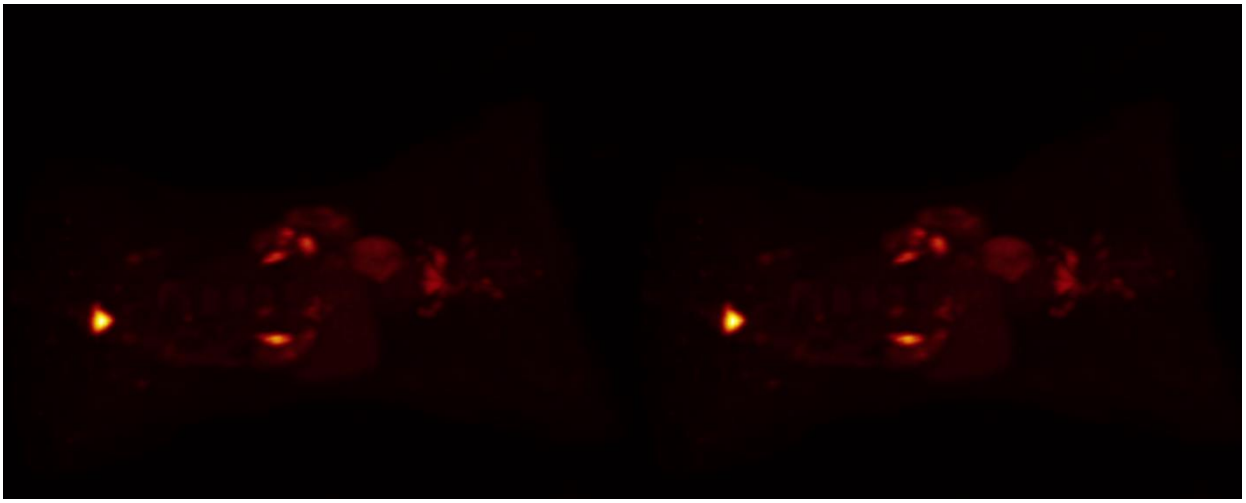

Рис. 4.4.2 Изображение туловища человека (PET снимки).

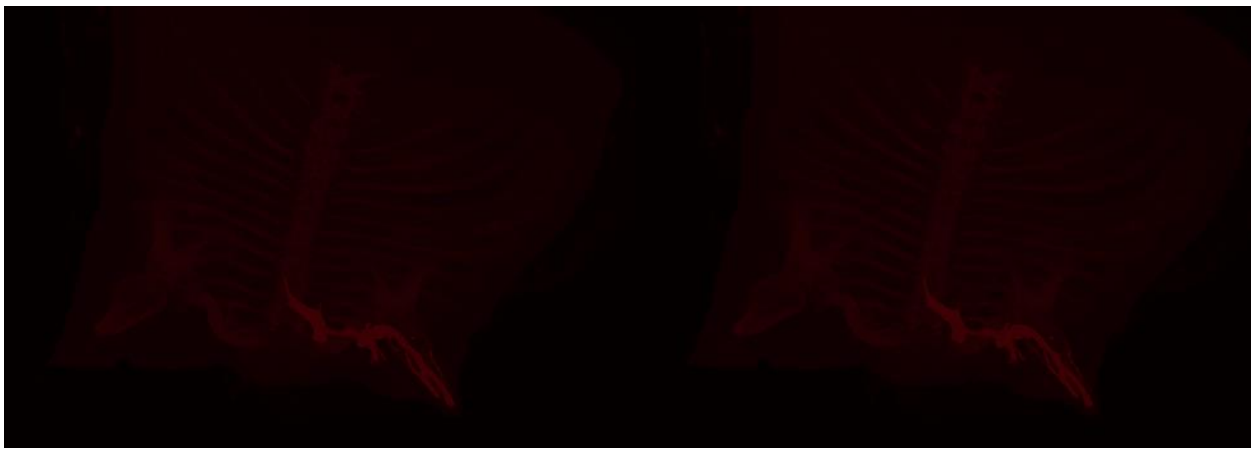

Рис. 4.4.3 Изображение туловища человека 1-й набор снимков(CT)

Рис. 4.4.4 Изображение туловища человека 2-й набор снимков(CT)

Глава 5. Тестирование

Для того, чтобы проверить корректность работы программного обеспечения, необходимо было уделить время ручному тестированию.

Во-первых, важно было проверить, правильно ли будет работать орбитальная камера и правильно ли будет происходить изменение положения объекта в пространстве, одновременно с этим проверяя управление с геймпада. Понять верно ли изменяются объемы, палитры, яркость и контрастность (таблица 5.1).

Тестовый случай	Ожидаемый результат
Вращение правого стика вправо/влево/вверх/вниз	Вращение объекта вправо/влево/вверх/вниз
Вращение левого стика вправо/влево/вверх/вниз	Перемещение объекта вправо/влево/вверх/вниз относительно плоскости xy
Нажать клавиши start несколько раз	Смена палитр по очереди на каждое нажатие
Нажать клавиши start и back	Сначала палитра меняется на следующую по списку, после нажатия back происходит возврат к прошлой палитре.
Изменить положение объекта в пространстве, а затем нажать B	Возврат к первоначальному положению объекта.
Нажать несколько раз Y	Изменяются объемы по очереди при каждом нажатии
Нажать Y, а затем X	При нажатии Y загружается следующий объем, после X происходит возврат к предыдущему объему.

Нажать вверх/вниз	Увеличение/уменьшение контрастности
Нажать вправо/влево	Увеличение/уменьшение яркости
Нажать RB/ LB	Приближение/удаление камеры
Нажать RT/LT	Вращение вправо/влево вокруг оси z

Таблица 5.1 Пример тест кейсов для проверки управления камерой, изменением контраста, яркости, палитр и объемов.

Во-вторых, проверить правильно ли считывается метаинформация, корректно ли строятся трехмерные объемные изображения (таблица 5.2).

Тестовый случай	Ожидаемый результат
Задан текстовый файл braininfo.txt, хранящий набор метаинформации и файл с изображениями brain.raw	На экране должно получиться изображение мозга, как на картинке 4.4.1
Изменить в текстовом файле braininfo.txt размер воксела по x в 2 раза.	На экране изображение растянется в 2 раза в ширину.
Изменить в текстовом файле braininfo.txt вектор ориентации y на противоположный.	У изображения меняются местами перед и зад.

Таблица 5.2 Пример тест кейсов для проверки построения трехмерных объемных изображений.

Для проверки работы готовой сборки было проведено конфигурационное тестирование. Важно убедиться, что приложение работоспособно на различных устройствах с разным набором обеспечения. Приложение было запущено на следующих устройствах:

1) Ноутбук MSI, процессор Intel Core i7, видеокарта NVIDIA GeForce GTX 870M

2) Стационарный компьютер, процессор Intel Core2Duo, видеокарта AMD Radeon HD7850
3) Ноутбук Lenovo, процессор Intel Core i5, видеокарта Nvidia GeForce GT 720

Тестовое окружение: очки виртуальной реальности Sony PlayStation VR, геймпад Xbox 360.

Далее приведен один из вариантов тестового сценария, который необходим на этапе системного тестирования, чтобы верифицировать весь имеющийся функционал в полученной сборке.

Тестовый сценарий:

1. Запустить приложение raytracing.exe с подключенными очками и геймпадом.
2. Повернуть изображение в левую сторону.
3. Сменить палитру.
4. Повернуть изображение вниз.
5. Изменить объем.
6. Изменить палитру.
7. Изменить яркость и контраст.
8. Повернуть объект вокруг оси z.
9. Приблизить и отдалить объект.
10. Изменить положение объекта в пространстве относительно плоскости xy.
11. Повернуть объект вправо.
12. Спросить к первоначальному положению.

Вывод

Написанное программное обеспечение удовлетворяет поставленным требованиям и в свою очередь, обладает ограниченным набором функций. В дальнейшем возможно улучшение качества программного обеспечения, посредством повышения производительности, добавления ряда новых функций, таких как управление камерой при помощи шлема виртуальной реальности, используя в основе показания датчиков движения головой, реализация совмещения различных изображений, полученных, например, при помощи CT и MRT, добавления нескольких методов визуализации объема для решения узконаправленных задач и повышения производительности. На сегодняшний день, работу можно использовать в демонстрационных целях в научных и учебных заведениях. Со временем работа будет дорабатываться и совершенствоваться, чтобы стать готовым продуктом.

Заключение

В ходе данной работы была выполнена поставленная цель о реализации программного обеспечения, которое визуализирует медицинские данные средствами виртуальной реальности. Главные особенности: работа с медицинскими данными формата dicom и raw data, получение объемной визуализации, построение стереоизображения, управление палитрами, яркостью и контрастностью, управление орбитальной камерой при помощи геймпада и мыши. Основное управление принадлежит геймпаду, поскольку именно так удобнее управлять, используя очки виртуальной реальности. В программе также существует возможность переключения между нескольким объемами. Работоспособность была проверена при помощи геймпада Xbox 360 и очков виртуальной реальности Sony PlayStation VR, программное обеспечение было протестировано вручную.

Список литературы

1) Медицинская визуализация, https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B4%D0%B8%D1%86%D0%B8%D0%BD%D1%81%D0%BA%D0%B0%D1%8F_%D0%B2%D0%B8%D0%B7%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F
2) Гаврилов Н.И. «Высокопроизводительная визуализация и морфологических анализ трехмерных данных в медицине и биологии», 2013, диссертация
3) Якименко К.Н. «Виртуальная реальность», Самиздат журнал, 2006
4) Что такое виртуальная реальность: свойства, классификация, оборудование – подробный обзор области, https://tproger.ru/translations/vr-explained/
5) Рендеринг, https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BD%D0%B4%D0%B5%D1%80%D0%B8%D0%BD%D0%B3
6) Андреас Поммерт,Бернард Пфлессер, Мартин Риемер, Томас Шиеманн, Райнер Шуберт, Вульф Тиеде, Карл Хейнц Хон «Визуализация объема в медицине»/ «Открытые системы.СУБД» выпуск №05,1996
7) Kaufman A. «Volume Visualization»/IEEE Computer Society Press Tutorial, 1991
8) Системы координат, https://habr.com/post/324968/
9) World,View and Projection Transformaion Matrices http://www.codinglabs.net/article_world_view_projection_matrix.aspx
10) OpenGL Transformation, http://www.songho.ca/opengl/gl_transform.html
11) Каверзные кватернионы, https://habr.com/post/183908/
12) Мищенко А., Соловьев Ю. «Кватернионы», Квант № 9,1983
13) GLSL Графический конвейер, http://savepearlharbor.com/?p=164065

14) Графический конвейер, http://elanina.narod.ru/lanina/ind/graph/file103.htm
15) OpenGL шейдеры. Простой шейдер GLSL http://esate.ru/uroki/OpenGL/uroki_opengl/_p4208/
16) OpenGL. Шейдеры. http://startandroid.ru/ru/uroki/vse-uroki-spiskom/398-urok-169-opengl-shejdery.html
17) Вращение объектов с помощью класса ArcBall, http://pmg.org.ru/nehe/nehe48.htm
18) Modern OpenGL Tutorial Arcball, https://en.wikibooks.org/wiki/OpenGL_Programming/Modern_OpenGL_Tutorial_Arcball
19) Проекции интенсивностей, http://www.kievoncology.com/proektsii-intensivnosti.html
20) Hashemi R. H., Bradley W. G., Lisanti C.J. «MRI: The Basics», Wolters Kluwer Health, 2012
21) Yoshinobu Sato, Nobuyuki Shiraga, Shin Nakajima, Shinchi Tamura, Ron Kikinis «Local Maximum Intensity Projection: A new rendering Method for Vascular Visualization»/ Journal of Computer Assisted Tomography №6, 1998
22) Elliot K. Fishman, Derek R. Ney, David G. Heath, Frank M. Corl, Karen M. Horton, Pamela T.Johnson «Volume rendering versus maximum intensity projection in CT Angiography», RadioGraphics, 2006
23) Dicom стандарт, http://www.course-as.ru/dicomdoc.html
24) Digia,Qt Learning «OpenGL Tutorial» Release 1.0, 2013
25) Данные, https://idoimaging.com/

Приложение 1

Схема управления геймпадом